



Clase 6. Python

# ***Conjuntos y Diccionarios***

***RECUERDA PONER A GRABAR LA  
CLASE***





## ***OBJETIVOS DE LA CLASE***

- Identificar un Conjunto
- Reconocer similitudes y diferencias entre list y set
- Agregar y borrar valores al set
- Identificar un Diccionario
- Agregar y borrar valores al dict

# ***CRONOGRAMA DEL CURSO***

## Clase 5



### **Controladores de flujo 2**



NÚMEROS



EJERCICIOS



CONTROL DE FLUJO

## Clase 6



### **Conjuntos y Diccionarios**



SETS



DICTS

## Clase 7



### **Métodos de colecciones**



COLECCIONES 1



COLECCIONES 2

***CONJUNTOS***

# ***¿Qué son?***

Un conjunto o **set** es una **colección no ordenada de objetos únicos**, es decir, no tiene elementos duplicados. Python provee este tipo de datos por defecto al igual que otras colecciones más convencionales como las **listas, tuplas y diccionarios**.



**¡Para recordar!**

Los conjuntos son ampliamente utilizados en lógica y matemática, y desde el lenguaje podemos sacar provecho de sus propiedades para crear código más eficiente y legible en menos tiempo.



# ***Conjuntos en python***



El conjunto se describe como una lista de ítems separados por coma y contenido entre dos llaves.

```
>>> conjunto = {1, 2, 3, 4}
>>> otro_conjunto = {"Hola", "como", "estas", "?"}
>>> conjunto_vacio = set()  #{ } [ ( )]
```

Para crear un conjunto vacío debemos decirle **set()** de lo contrario si quisiéramos hacer como las listas y crearlo con {} python crea un diccionario, el cual veremos más adelante 🙏

***CODER HOUSE***



***HETEROGÉNEOS***

# ***Heterogéneos***



En otros lenguajes, las colecciones tienen una restricción la cual sólo permite tener un sólo tipo de dato. Pero en Python, no tenemos esa restricción. Podemos tener un **conjunto heterogéneo** que contenga números, variables, strings, o tuplas.

Ejemplo:

```
>>> mi_var = 'Una variable'
>>> datos = {1, -5, 123.1, 34.32, 'Una cadena', 'Otra cadena',
mi_var}
```



# ***Heterogéneos***

Sin embargo, un conjunto **no** puede incluir objetos mutables como listas, diccionarios, e incluso otros conjuntos o **set**.

Ejemplo:

```
>>> s = {{1,2}, [1,2,3,4],2}
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
TypeError: unhashable type: 'set'
```

# ***Heterogéneos***



De la misma forma podemos obtener un conjunto a partir de cualquier objeto

**iterable:**

```
>>> set1 = set([1, 2, 3, 4])  
{1, 2, 3, 4}  
>>> set2 = set(range(10))  
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

***SET***

# Set



Un set puede ser **convertido** a una **lista** y **viceversa**. En este **último** caso, los elementos **duplicados** son **unificados**.

```
>>> list({1, 2, 3, 4})  
[1, 2, 3, 4]  
>>> set([1, 1, 2, 2, 3, 3, 4, 4])  
{1, 2, 3, 4}
```

# List vs. Set



Ejemplo:

```
>>> conjunto = {'a', 'b', 'c', 'd', 'e', 'f'}
```

```
>>> conjunto[:3] = ['A', 'B', 'C']
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'set' object is not  
subscriptable

Como hablamos, las listas son **mutables**, sin embargo, el set también es **mutable**, pero no podemos hacer slicing, ni manejar un set por índice.

# ***FUNCIONES DE CONJUNTOS***



# ***Funciones Integradas***

En los conjuntos, hay funciones que son muy interesantes e importantes, las **funciones integradas.**

Los conjuntos en python tienen muchas funciones para utilizar, entre todas ellas vamos a nombrar las más importantes.

***ADD***

# Add



La primer función de los conjuntos de la que estaremos hablando es **ADD**. Esta función **permite agregar un nuevo ítem al set**. La misma se escribe

`mi_conjunto.add(ítem_a_agregar)`

Ejemplo:

```
>>> numeros = {1,2,3,4}
>>> numeros.add(5)
{1,2,3,4,5}
```

**mi\_conjunto** sería el set al que se le desee agregar el ítem, e **ítem\_a\_agregar** sería el ítem que deseemos agregar al set.

# Add



Ejemplo:

```
>>> numeros = {1,2,3,4}
>>> numeros.add(3*2)
{1,2,3,4,6}
>>> numeros.add(3**2+1-12+5*)
{1,2,3,4,6,13}
```

No sólo acaba ahí. En la función add también podemos realizar operaciones aritméticas en nuestro ítem.

***UPDATE***

# Update



Para añadir múltiples elementos a un set se usa la función **update()**, que puede tomar como argumento una lista, tupla, string, conjunto o cualquier objeto de tipo iterable.

La misma se escribe:

```
mi_conjunto.update(item_a_agregar)
```

```
>>> numeros = {1,2,3,4}
>>> numeros.update([5,6,7,8])
{1,2,3,4,5,6,7,8}
>>> numeros.update(range(9,12))
{1,2,3,4,5,6,7,8,9,10,11}
```

***LEN***

# Longitud del set



¿Se acuerdan cuando  
hablamos de **len** en listas?

En set, se puede usar  
exactamente la misma función  
para poder saber la longitud de  
un set, es decir, la cantidad de  
ítems dentro del mismo.

Ejemplo:

```
>>> numeros = {1,2,3,4}
```

```
>>> len(numeros)
```

```
4
```

```
>>> datos = {1, -5, 123.34, 'Una cadena', 'Otra cadena'}
```

```
>>> len(datos)
```

```
5
```



***DISCARD***

# Discard



Si **add** te deja agregar un ítem al set, **discard** hace todo lo contrario, **elimina el ítem del set**, sin modificar el resto del set, si el elemento pasado como argumento a **discard()** no está dentro del conjunto es simplemente ignorado.

```
>>> numeros = {1, 2, 3, 4}
>>> numeros.discard(2)
{1, 3, 4}
>>> datos = {1, -5, 123,34, 'Una cadena', 'Otra cadena'}
>>> datos.discard('Otra cadena')
{1, -5, 123,34, 'Una cadena'}
```

Se escribe como  
**mi\_conjunto.discard(item\_a\_descartar).**

***REMOVE***

# Remove



La función **remove** funciona igual al **discard**, pero con una diferencia, en **discard** si el ítem a remover **no existe**, simplemente se ignora. En **remove** en este caso nos **indica un error**.

```
>>> numeros = {1, 2, 3, 4}
```

```
>>> numeros.remove(2)
```

```
{1, 3, 4}
```

```
>>> numeros.remove(5)
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
KeyError: 5
```

Se escribe como `mi_conjunto.remove(item_a_remove)`.

***IN***

# ***In***



Para determinar si un elemento **pertenece** a un set, utilizamos la palabra reservada **in**.

Se escribe como **item\_a\_validar in mi\_conjunto**

```
>>> numeros = {1, 2, 3, 4}
>>> 2 in numeros
True
>>> 2 not in numeros
False
>>> 4 in numeros
False
```

***CLEAR***

# *Clear*



Igual que en las listas, podremos borrar todos los valores de un set simplemente usando la función **clear**.

Se escribe como `mi_conjunto.clear()`.

```
>>> numeros = {1, 2, 3, 4}
>>> numeros.clear()
set()
```



¡No se puede asignar un set vacío por que lo toma como diccionario!



***POP***

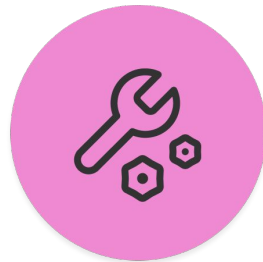


# Pop

La función **pop** retorna un elemento en forma aleatoria (no podría ser de otra manera ya que los elementos no están ordenados).

Así, el siguiente bucle imprime y remueve uno por uno los miembros de un conjunto.

```
>>> numeros = {1,2,3,4}
>>> while numeros:
    print("Se está borrando: ", numeros.pop())
Se está borrando: 1
Se está borrando: 2
Se está borrando: 3
Se está borrando: 4
```



# ***SETS***

Programa las instrucciones sobre la variable grupo

Tiempo estimado: 10 minutos



# ***DESAFÍO DE SETS***

Tiempo estimado: 10 minutos

Programa las siguientes instrucciones de forma ordenada sobre la variable **grupo**:

1. Añade los usuarios: **Ana, Ramón, Marta, Eric, David**
2. Elimina los usuarios: **Mario, Miguel, Esteban**

```
grupo = {"Miguel", "Blanca", "Mario", "Andrés"}
```



***BREAK***

**¡5/10 MINUTOS Y VOLVEMOS!**

# ***DICCIONARIOS***



# *¿Qué son?*

Un diccionario **dict** es una colección no ordenada de objetos. Es por eso que para identificar un valor cualquiera dentro de él, especificamos una **clave** (a diferencia de las listas y tuplas, cuyos elementos se identifican por su posición).

Las **claves** suelen ser **int** o **string**, aunque cualquier otro objeto inmutable puede actuar como una clave. Los valores, por el contrario, pueden ser de cualquier tipo, incluso otros diccionarios.



# *¿Cómo se crean?*

Para crear un diccionario se emplean llaves {}, y sus pares clave-valor se separan por comas. A su vez, intercalamos la clave del valor con dos puntos (:)

```
>>> colores = {"amarillo": "yellow", "azul": "blue", "rojo": "red"}  
{"amarillo": "yellow", "azul": "blue", "rojo": "red"}  
>>> type(colores)  
<class 'dict'>
```

**Nota:** Para crear un diccionario vacío se puede hacer `diccionario = {}`





# ***¿Cómo traer valor de Diccionarios?***



```
>>> colores = {"amarillo": "yellow", "azul": "blue", "rojo": "red"}
>>> colores["amarillo"]
"yellow"
>>> colores["azul"]
"blue"
>>> numeros = {10:"diez", 20:"veinte"}
>>> numeros[10]
"diez"
```

Para traer el  
valor de un  
**diccionario** se  
utiliza su **clave**

***MUTABILIDAD***



# ***Mutabilidad***

Los diccionarios al igual que las listas son **mutables**, es decir, que podemos reasignar sus ítems haciendo referencia con el índice.

```
>>> colores = {"amarillo": "yellow", "azul": "blue", "rojo": "red"}  
>>> colores["amarillo"] = "white"  
>>> colores["amarillo"]  
"white"
```



# ***Asignación***



```
>>> edades = {"Juan": 26, "Esteban": 35, "Maria": 29}
>>> edades["Juan"] += 5
>>> edades["Juan"]
31
>>> edades["Maria"] *= 2
>>> edades["Maria"]
58
```

También permite  
operaciones en  
asignación

# ***FUNCIONES DE DICCIONARIOS***

# ***Funciones de Diccionarios***

Al igual que en conjuntos, en los diccionarios encontramos **funciones integradas.**

Los diccionarios en python tienen muchas funciones para utilizar. Si bien hablaremos las desarrollaremos más adelante, a continuación vamos a nombrar las más importantes.

***ADD***

# Add



**No hay una función de add**, pero para agregar una nueva clave-valor se puede realizar de la siguiente manera:

```
>>> numeros = {"uno": 1, "dos": 2, "tres": 3, "cuatro": 4}
>>> numeros["cinco"] = 5
{"uno": 1, "dos": 2, "tres": 3, "cuatro": 4, "cinco": 5}
```

En este caso, creamos una nueva clave que no existe "**cinco**" y asignamos el valor **5**



***UPDATE***

# Update



Este método actualiza un diccionario agregando los pares clave-valores.

```
>>> numeros = {"uno": 1, "dos": 2, "tres": 3, "cuatro": 4}
>>> numeros.update({"cinco": 5, "seis": 6})
{"uno": 1, "dos": 2, "tres": 3, "cuatro": 4, "cinco": 5, "seis": 6}
>>> otro_dict = dict(siete=7)
>>> numeros.update(otro_dict)
{"uno": 1, "dos": 2, "tres": 3, "cuatro": 4, "cinco": 5, "seis": 6, "siete": 7}
```

# ***Update***



El método `update()` **toma un diccionario o un objeto iterable de pares clave/valor (generalmente tuplas)**. Si se llama a `update()` sin pasar parámetros, el diccionario permanece sin cambios.

***LEN***



# ***Longitud del diccionario***

¿Se acuerdan cuando hablamos de **len** en listas? En dict, se puede usar exactamente la misma función para poder saber la longitud de un dict, es decir, la cantidad de ítems dentro del mismo.

```
>>> numeros = {"uno": 1, "dos": 2, "tres": 3, "cuatro": 4}
>>> len(numeros)
4
```

***DEL***

# ***Del***



**Del** elimina el ítem del dict, sin modificar el resto del dict, si el elemento pasado como argumento a **del()** no está dentro del dict es simplemente ignorado.

Se escribe como **del** mi\_dict[**“clave”**].

```
>>> numeros = {"uno": 1, "dos": 2, "tres": 3, "cuatro": 4}
>>> del numeros["dos"]
{"uno": 1, "tres": 3, "cuatro": 4}
```

***IN***



# In



Para determinar si un elemento **pertenece** a un dict, utilizamos la palabra reservada **in**.

```
>>> numeros = {"uno": 1, "dos": 2, "tres": 3, "cuatro": 4}
>>> "dos" in numeros
True
>>> 2 not in numeros
False
>>> 4 in numeros
False
```

Se escribe  
como  
clave\_a\_validar  
**in** mi\_dict

***CLEAR***



# ***Clear***

Igual que en las listas, podremos borrar todos los valores de un dict simplemente usando la función **clear**.

Se escribe como dict.**clear()**.

```
>>> numeros = {"uno": 1, "dos": 2, "tres": 3, "cuatro": 4}
>>> numeros.clear()
{}

```



Otra forma más cómoda es hacer `mi_dict = {}`

***CODER HOUSE***

***POP***

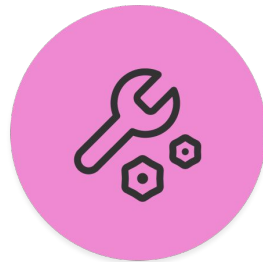


# Pop

Este método **remueve específicamente una clave de diccionario y devuelve valor correspondiente**. Lanza una excepción `KeyError` si la clave no es encontrada.

Se escribe como `mi_dict.pop("clave")`

```
>>> numeros = {"uno": 1, "dos": 2, "tres": 3, "cuatro": 4}
>>> numeros.pop("uno")
{"dos": 2, "tres": 3, "cuatro": 4}
```



# ***DICTS***

Programa las instrucciones sobre la variable animales

Tiempo estimado: 10 minutos



# DESAFÍO DE SETS

Tiempo estimado: 10 minutos

Programa las siguientes instrucciones de forma ordenada sobre la variable **animales**:

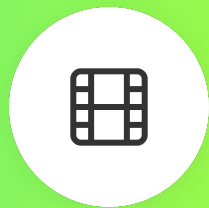
Inicialmente el diccionario es: **animales = {"elefante": ""}**

1. Añade al diccionario las claves **perro**, **tigre** y **mono** con sus respectivos valores **"Bobby"**, **"Peepe"** y **"homero"**
2. Modificá las claves **elefante** y **delfin** con los valores **"Trompis"** y **"Manolo"** respectivamente

***¿PREGUNTAS?***







***¿QUIERES SABER MÁS? TE DEJAMOS  
MATERIAL AMPLIADO DE LA CLASE***



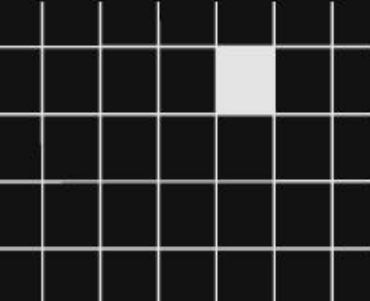
- Artículo: [Set y Funciones](#)
- Artículo: [Diccionarios y Funciones](#)
- [EjemploClase](#)





# ***¡MUCHAS GRACIAS!***

Resumen de lo visto en clase hoy:

- Listas
  - Tuplas
  - Anidación
  - Transformación de colecciones
- 



***OPINA Y VALORA ESTA CLASE***

***#DEMOCRATIZANDO LA EDUCACIÓN***

***CODER HOUSE***